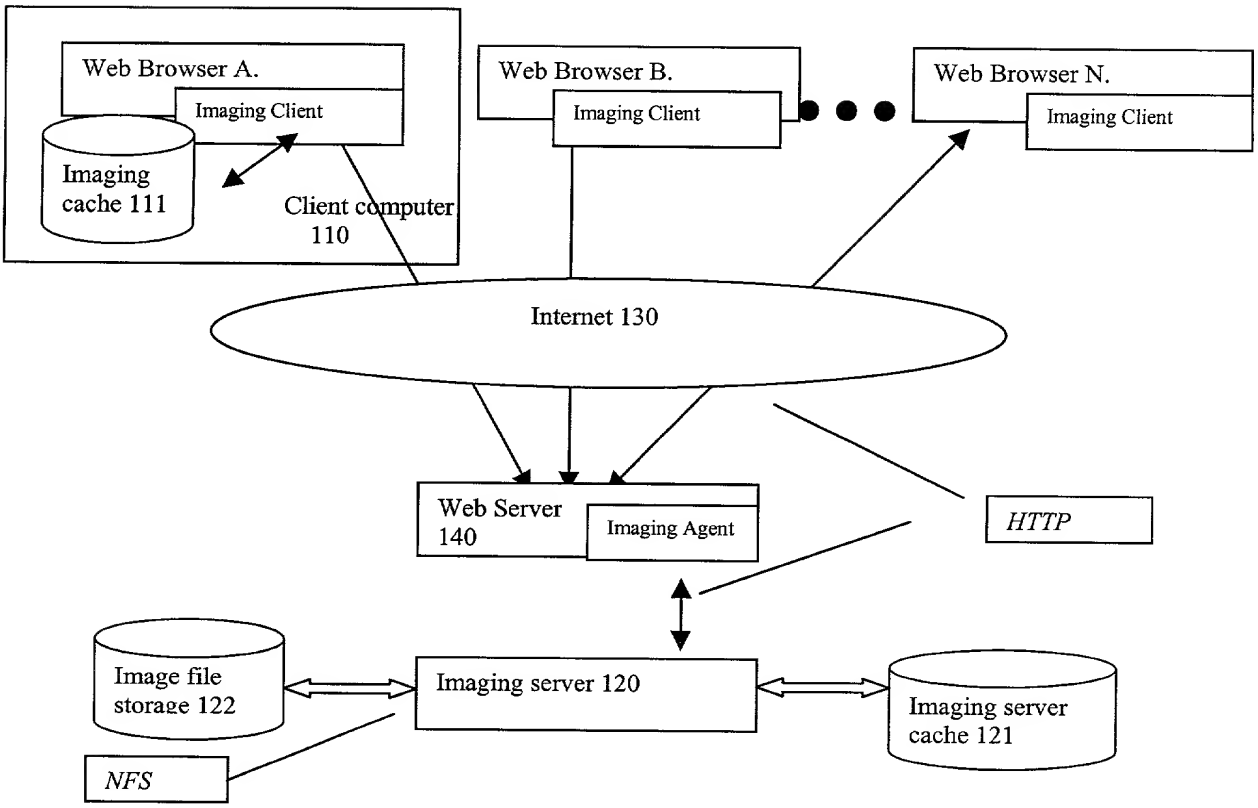


Figure 1 **System Architecture Block Diagram**

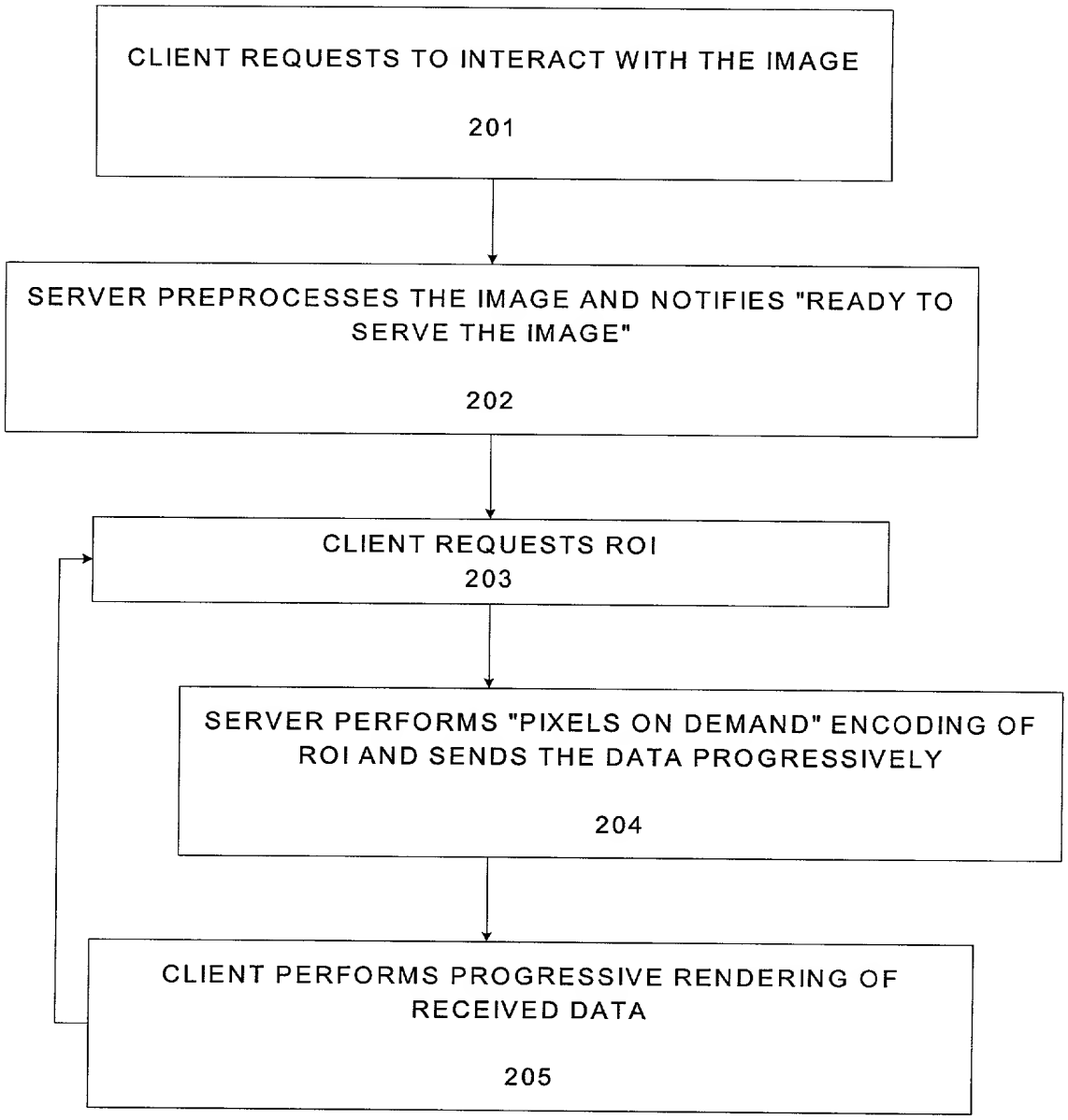

CLIENT REQUESTS TO INTERACT WITH THE IMAGE
201
SERVER PREPROCESSES THE IMAGE AND NOTIFIES "READY TO SERVE THE IMAGE"
202
CLIENT REQUESTS ROI
203
SERVER PERFORMS "PIXELS ON DEMAND" ENCODING OF ROI AND SENDS THE DATA PROGRESSIVELY
204
CLIENT PERFORMS PROGRESSIVE RENDERING OF RECEIVED DATA
205

FIG. 2

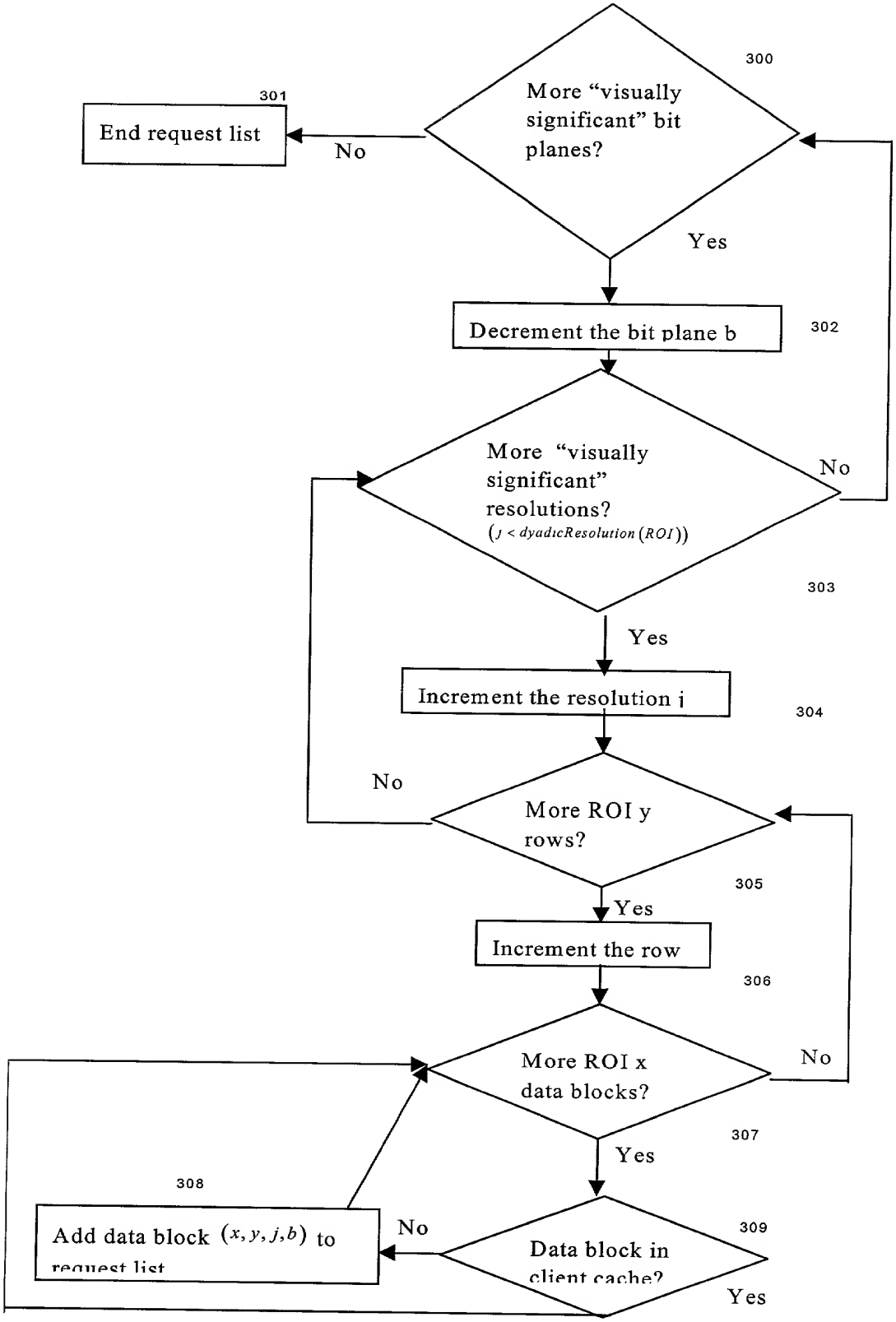

300
More "visually significant" bit planes?
301
End request list
No
Yes
Decrement the bit plane b
302
More "visually significant" resolutions?
$(j < dyadicResolution(ROI))$
No
303
Yes
Increment the resolution i
304
No
More ROI y rows?
305
Yes
Increment the row
306
More ROI x data blocks?
No
307
Yes
308
Add data block (x, y, j, b) to request list
No
309
Data block in client cache?
Yes

Figure 3

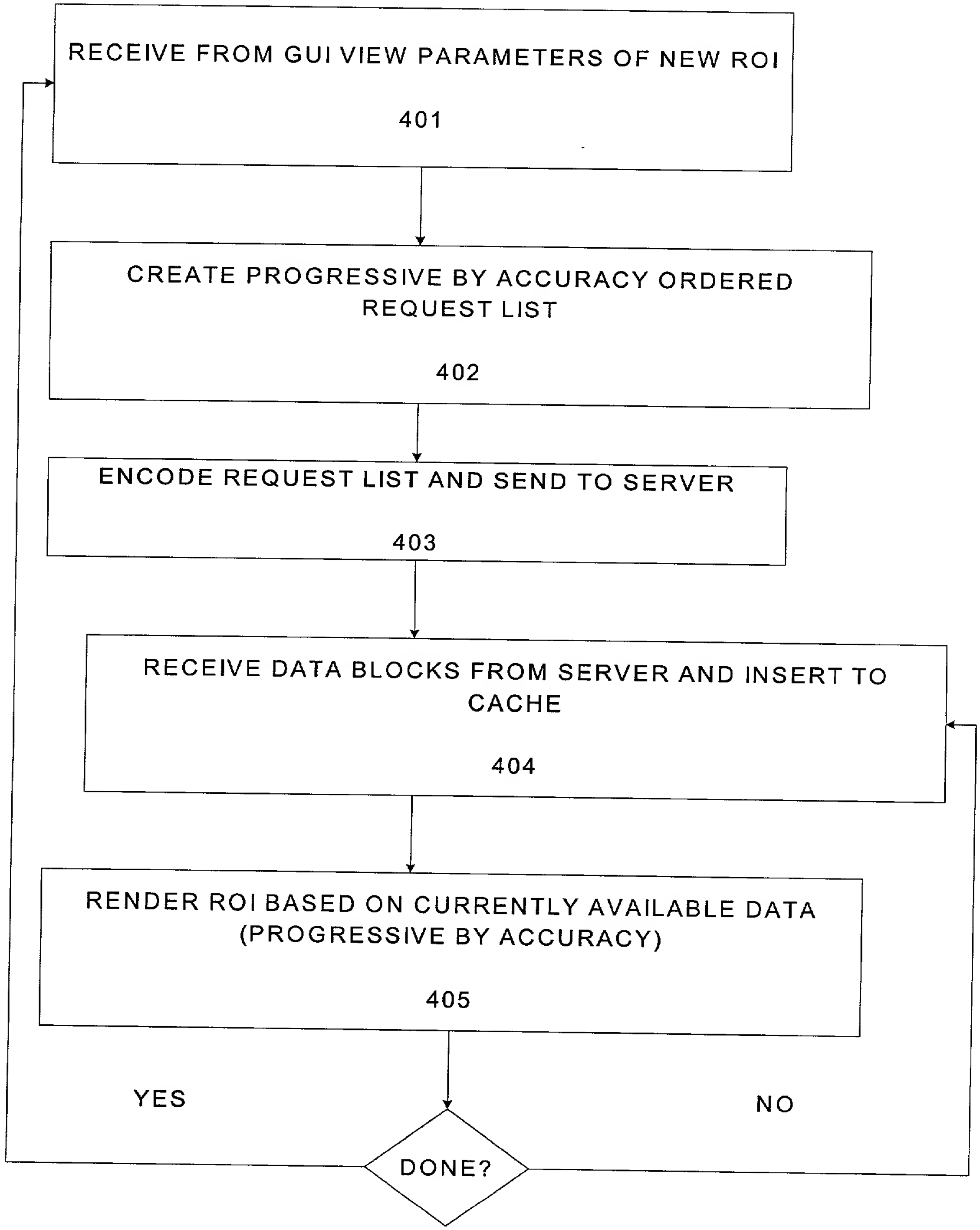

RECEIVE FROM GUI VIEW PARAMETERS OF NEW ROI
401
CREATE PROGRESSIVE BY ACCURACY ORDERED REQUEST LIST
402
ENCODE REQUEST LIST AND SEND TO SERVER
403
RECEIVE DATA BLOCKS FROM SERVER AND INSERT TO CACHE
404
RENDER ROI BASED ON CURRENTLY AVAILABLE DATA (PROGRESSIVE BY ACCURACY)
405
YES
NO
DONE?

FIG. 4

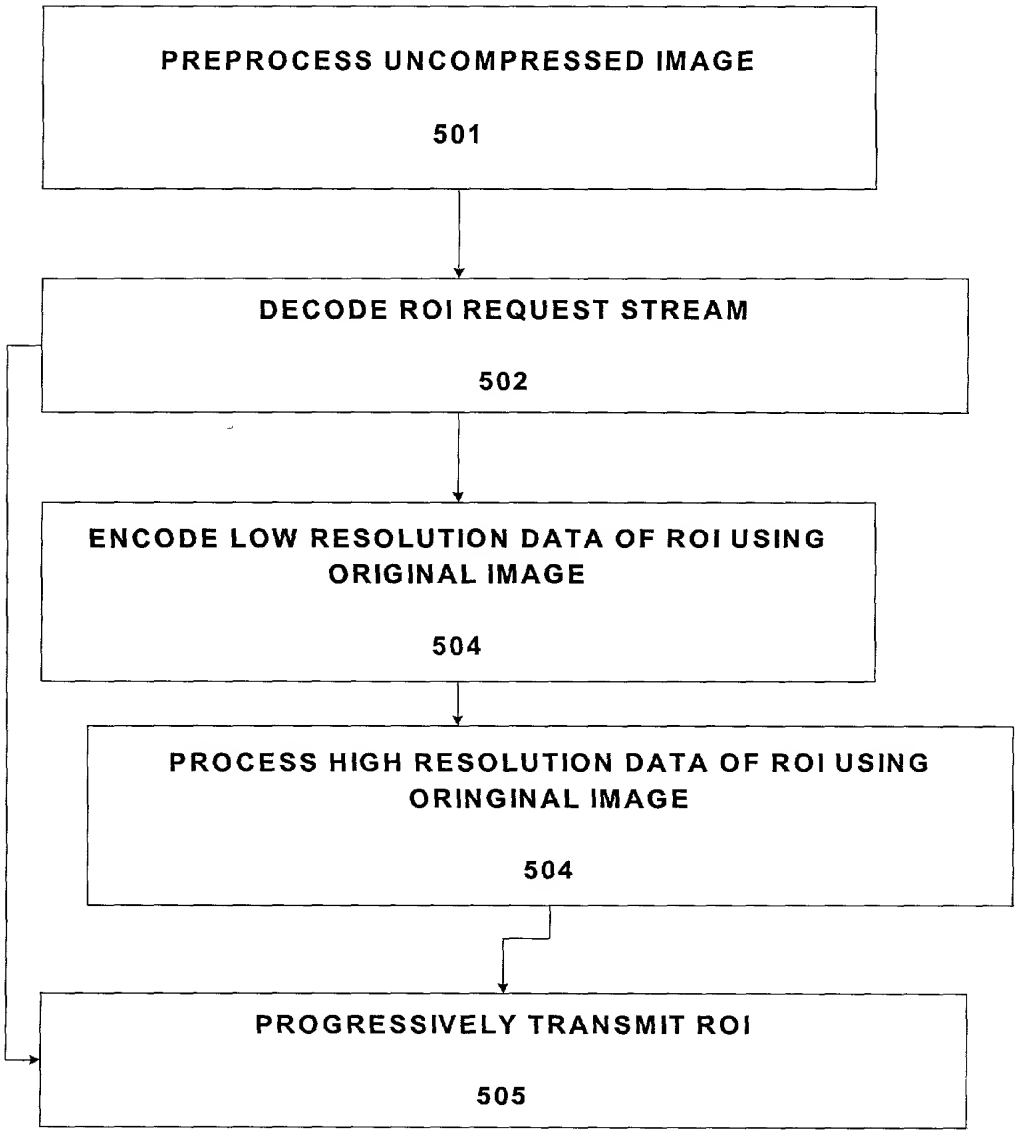

PREPROCESS UNCOMPRESSED IMAGE
501
DECODE ROI REQUEST STREAM
502
ENCODE LOW RESOLUTION DATA OF ROI USING ORIGINAL IMAGE
504
PROCESS HIGH RESOLUTION DATA OF ROI USING ORINGINAL IMAGE
504
PROGRESSIVELY TRANSMIT ROI
505

FIG. 5

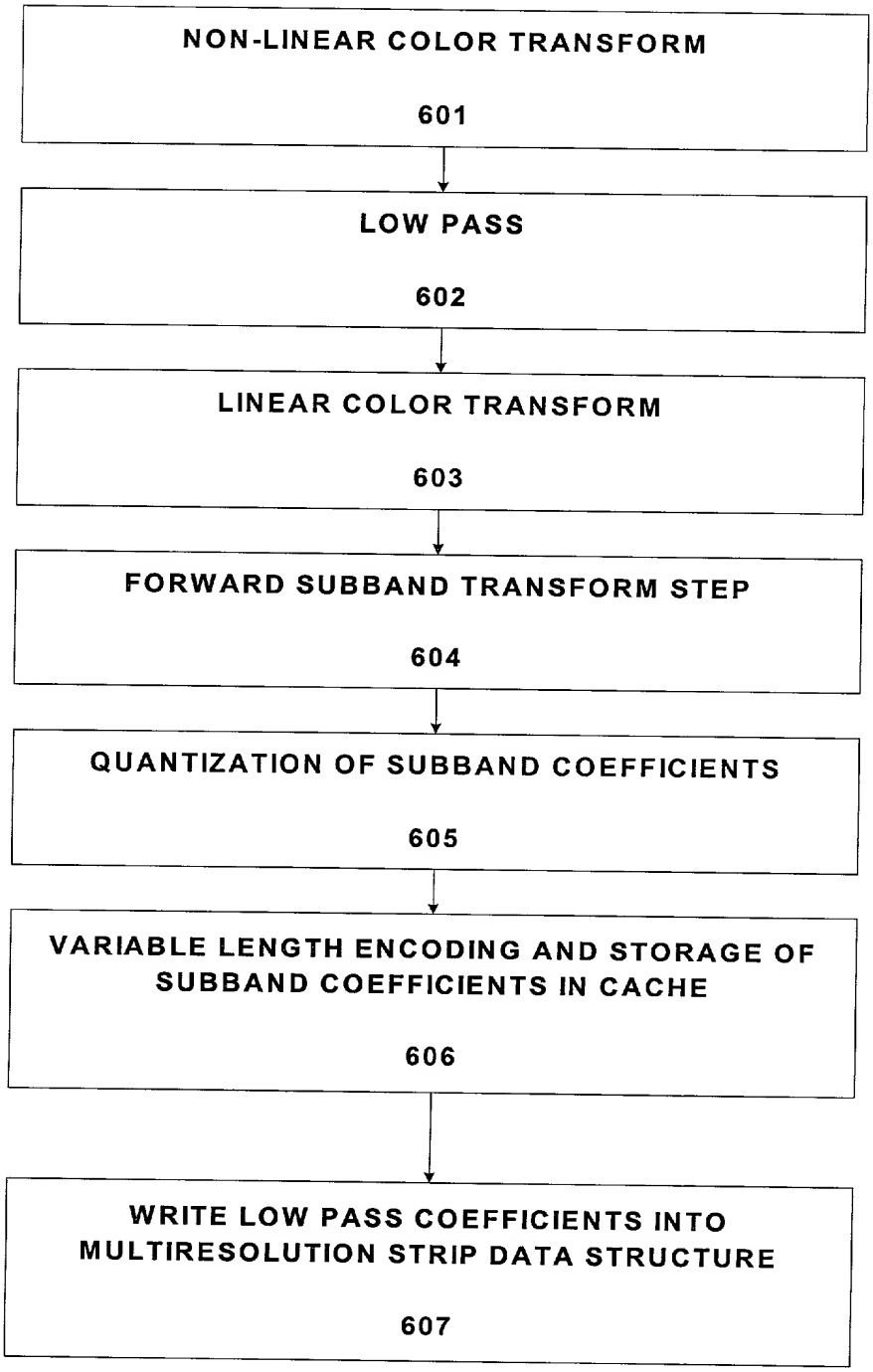

NON-LINEAR COLOR TRANSFORM
601
LOW PASS
602
LINEAR COLOR TRANSFORM
603
FORWARD SUBBAND TRANSFORM STEP
604
QUANTIZATION OF SUBBAND COEFFICIENTS
605
VARIABLE LENGTH ENCODING AND STORAGE OF SUBBAND COEFFICIENTS IN CACHE
606
WRITE LOW PASS COEFFICIENTS INTO MULTIRESOLUTION STRIP DATA STRUCTURE
607

FIG. 6

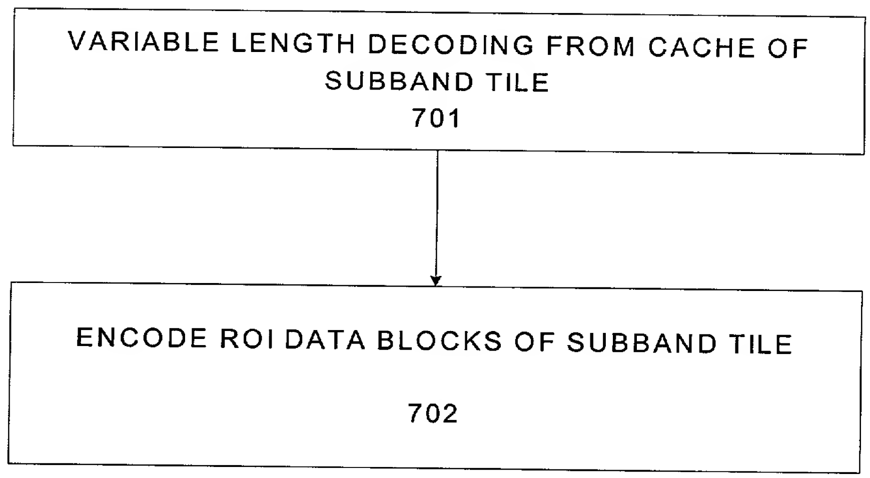

VARIABLE LENGTH DECODING FROM CACHE OF SUBBAND TILE
701
ENCODE ROI DATA BLOCKS OF SUBBAND TILE
702

FIG. 7

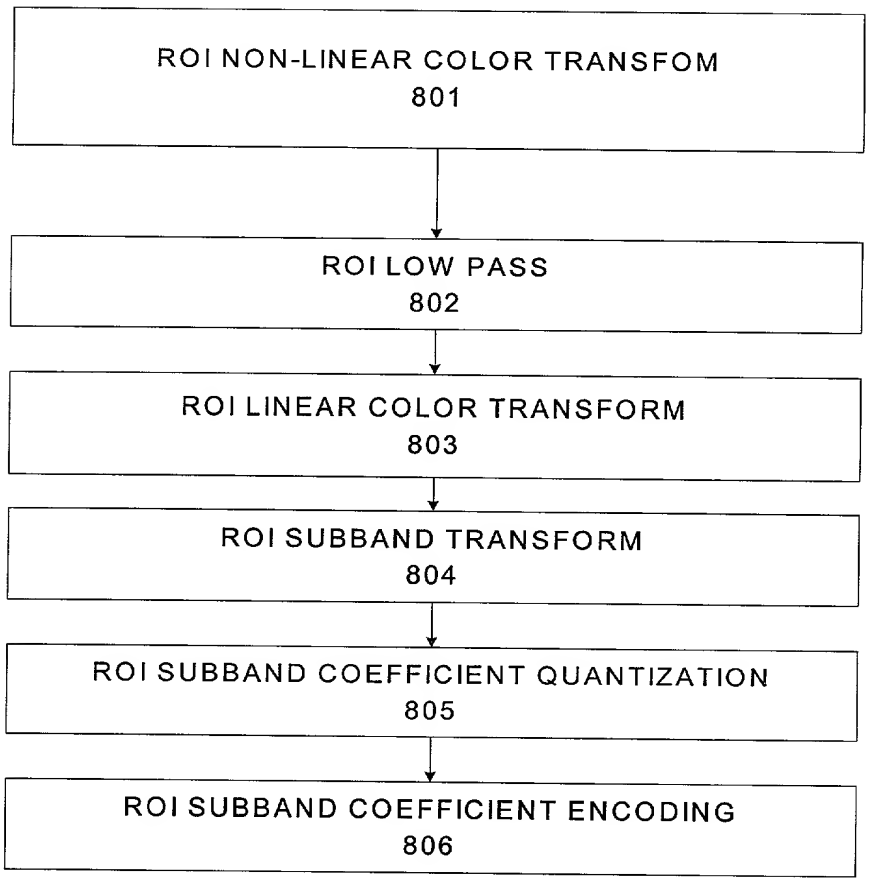

ROI NON-LINEAR COLOR TRANSFOM
801
ROI LOW PASS
802
ROI LINEAR COLOR TRANSFORM
803
ROI SUBBAND TRANSFORM
804
ROI SUBBAND COEFFICIENT QUANTIZATION
805
ROI SUBBAND COEFFICIENT ENCODING
806

FIG. 8

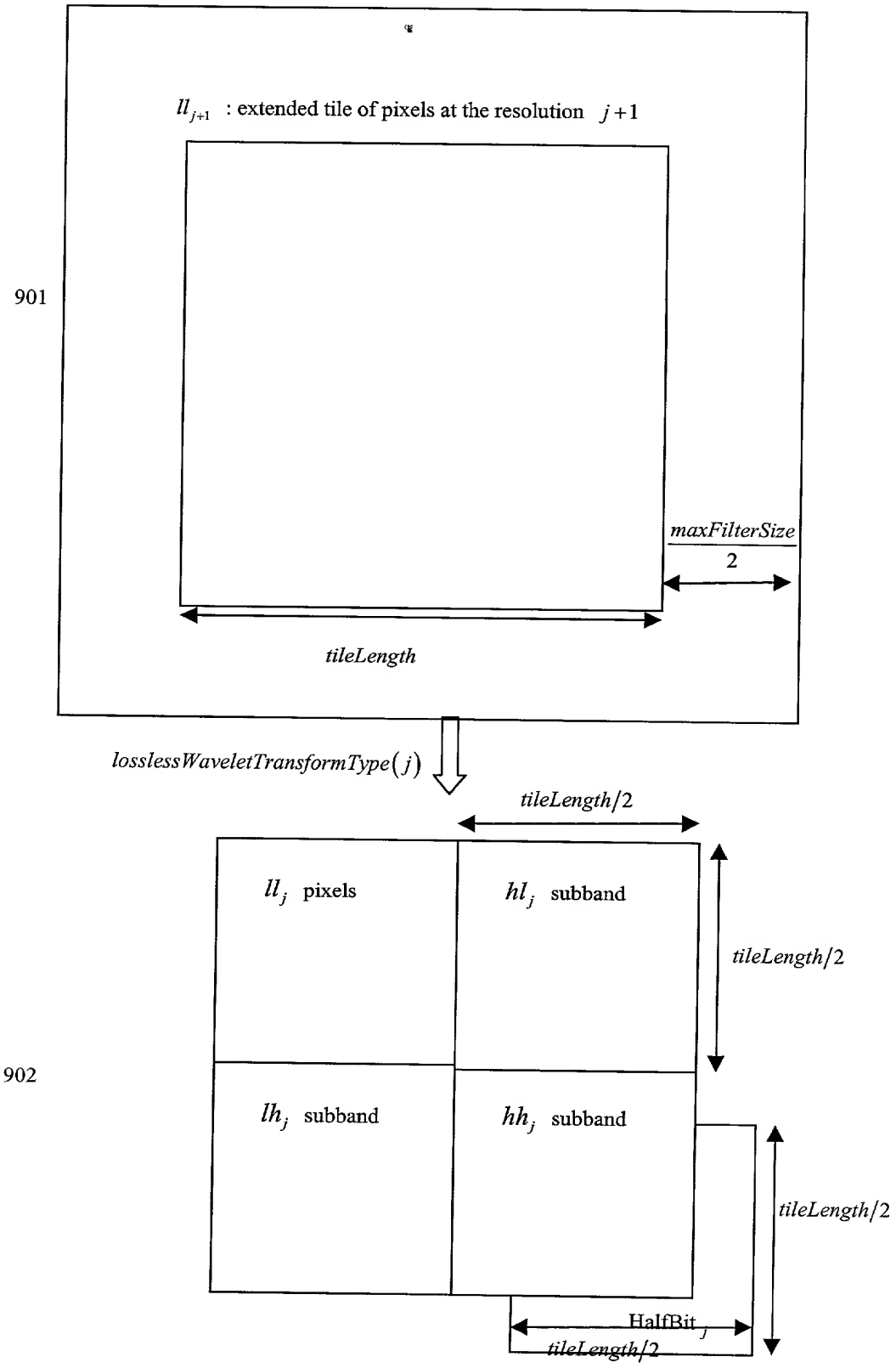

ll_{j+1} : extended tile of pixels at the resolution $j+1$
901
$\frac{maxFilterSize}{2}$
tileLength
losslessWaveletTransformType(j)
tileLength/2
ll_j pixels
hl_j subband
tileLength/2
902
lh_j subband
hh_j subband
tileLength/2
HalfBit $_j$
tileLength/2

Figure 9

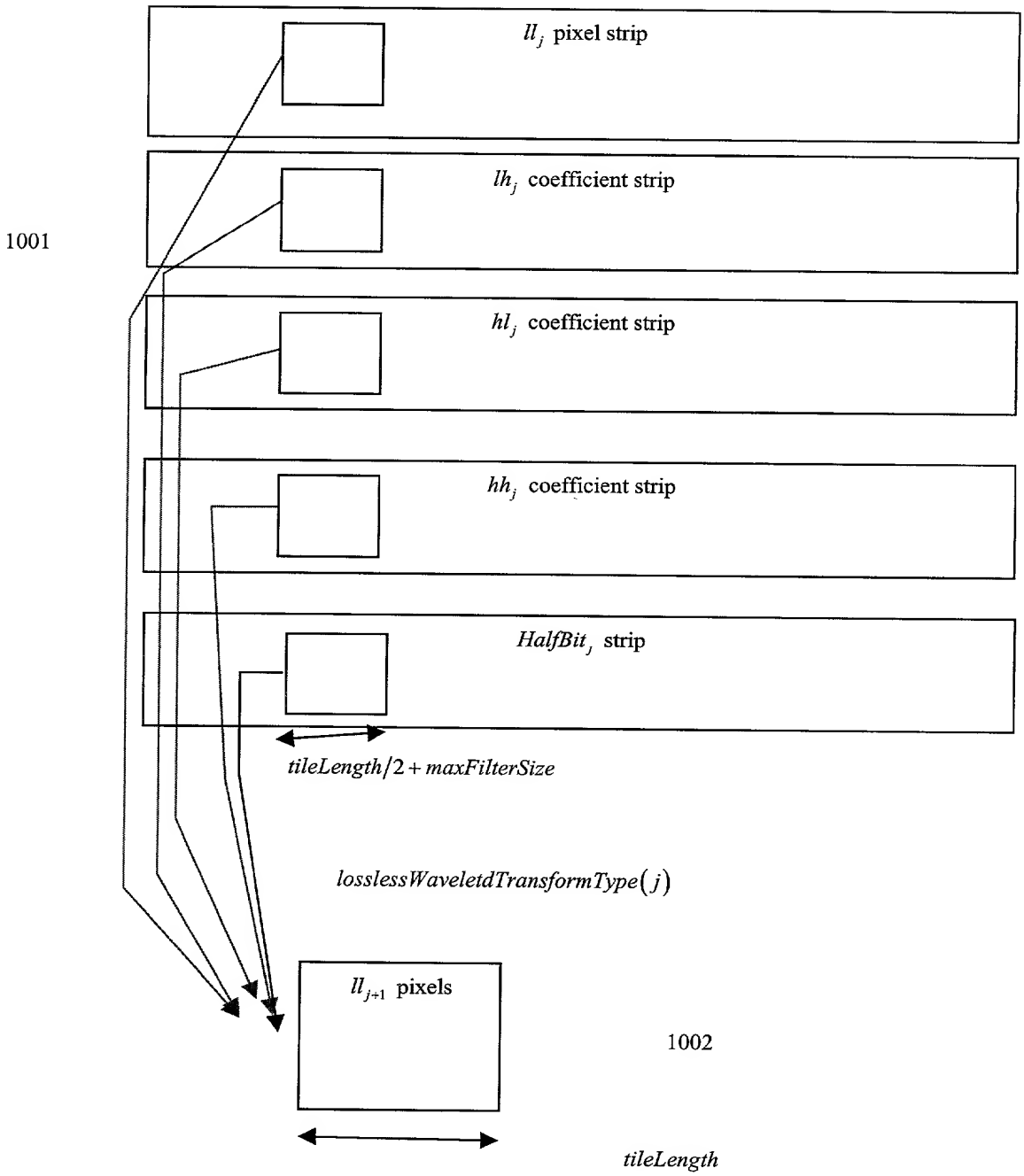

ll_j pixel strip
lh_j coefficient strip
hl_j coefficient strip
hh_j coefficient strip
$HalfBit_j$ strip
1001
$tileLength/2 + maxFilterSize$
$losslessWaveletdTransformType(j)$
ll_{j+1} pixels
1002
$tileLength$

Figure 10

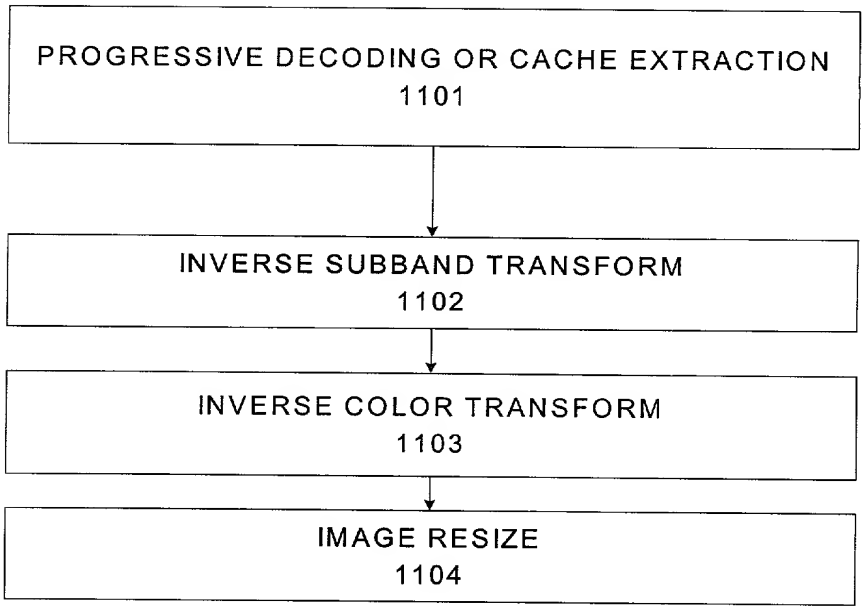


FIG. 11

1201

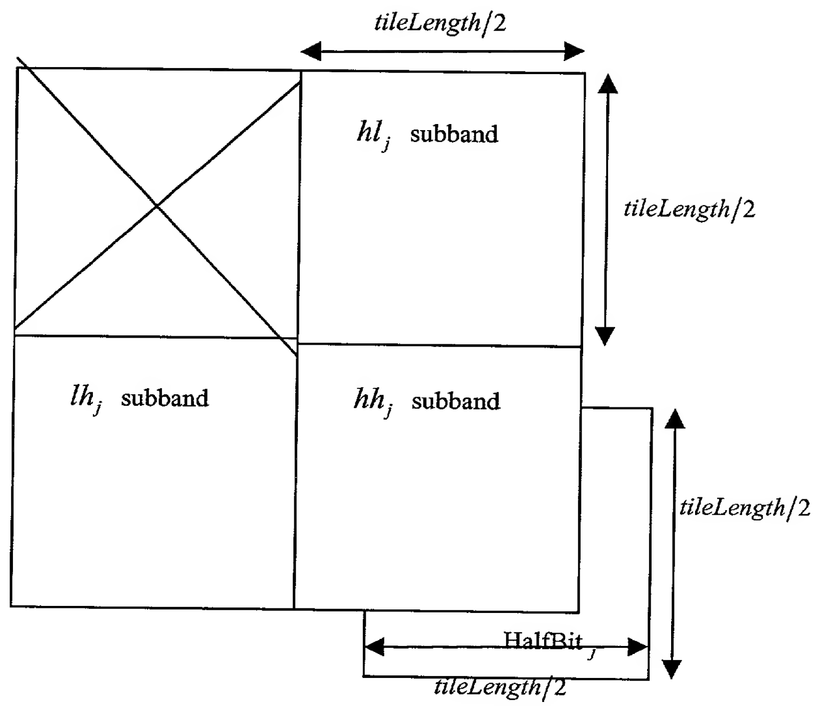


Figure 12

RGB <-> YUV Reversible Conversion

Forward:

$$Y_r = \left\lfloor \frac{R + 2G + B + 2}{4} \right\rfloor$$

$$U_r = R - G$$

$$V_r = B - G$$

Inverse:

1301

$$G = Y_r - \left\lfloor \frac{U_r + V_r + 2}{4} \right\rfloor$$

$$R = U_r + G$$

$$B = V_r + G$$

Figure 13

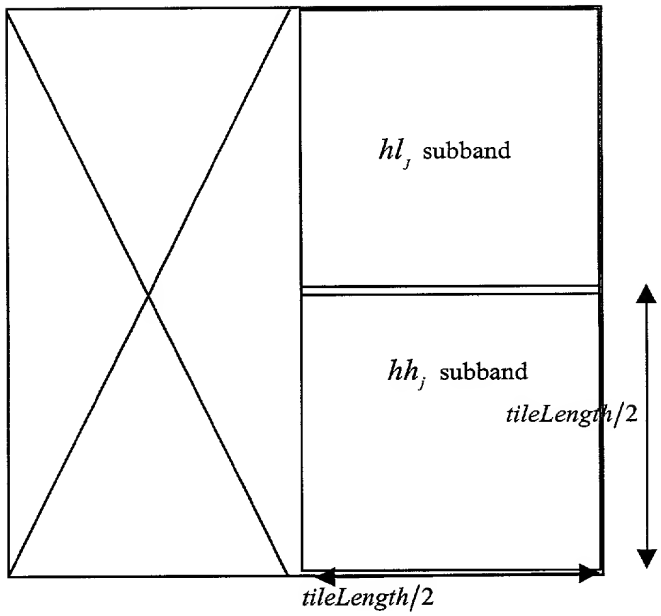


Figure 14

```
bitModel.startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();

while (encoder.moreCoef()) {
  if (encoder.isCoefReported()) {

arithmetic_encode_symbol(bitModel,encoder.reportedCoefPrec
isionBit());
  }
  else {
    if ( encoder.isCoefExactZero() );
     arithmetic_encode_symbol(zeroCoefModel,true);
    else {
     arithmetic_encode_symbol(zeroCoefModel,false);
     arithmetic_encode_symbol(coefSignModel,encoder.getCoe
fSign());
    }
  }
}
```

(a)

```
bitModel.startModel();

for (int i = 0 ; i < hBlockSize ; i++) {
     for (int j = 0 ; j < hBlockSize ; j++) {
          arithmetic_encode_symbol(bitModel,
coefHalfBit[i][j]);
     }
}
```

(b)

Figure 15

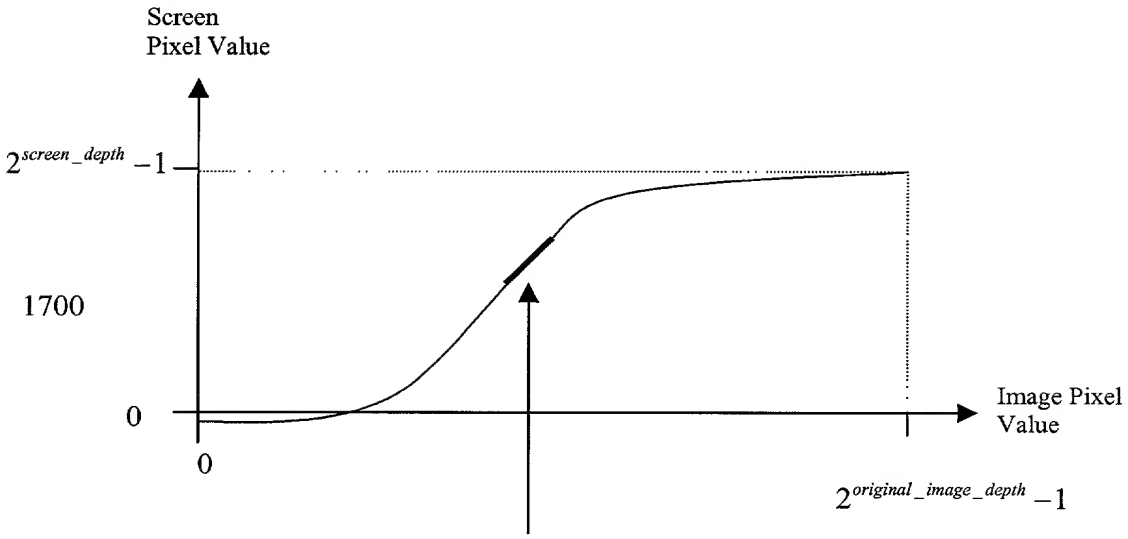
```
                    bitModel      .startModel();
zeroCoefModel.startModel();
coefSignModel.startModel();

decoder.initializeLSBPlaneCoefScan();

while (decoder.moreCoef()) {
  if (decoder.isCoefReported()) {
    if(decoder.isLHCoef()) {
      decoder. updateLSB (0);
    }
    else {
     decoder.updateLSB(arithmetic_decoder_symbol(bitModel));
    }
  }
  else {
    if(!decoder.isLHCoef()) {
      if (!arithmetic_decoder_symbol(zeroCoefModel))
         decoder.setLSB(arithmetic_decoder_symbol(coefSignMode
         l));
    }
  }
}
```

(a)

```
bitModel.startModel();
for (int i = 0 ; i < hBlockSize ; i++) {
  for (int j = hBlockSize ; j ; j--,p++) {
    coefHalfBit[i][j] = arithmetic_decoder_symbol(bitModel);
  }
}
```

(b)

Figure 16



Maximal gradient

1701

Figure 17

1800

$$s=\left[s(1),s(2),\ldots,s\left(\frac{N}{2}\right)\right]$$

1D Wavelet transform step

$$x=\left[x(1),x(2),\ldots,x(N)\right]\Longrightarrow$$

$$d=\left[d(1),d(2),\ldots,d\left(\frac{N}{2}\right)\right]$$

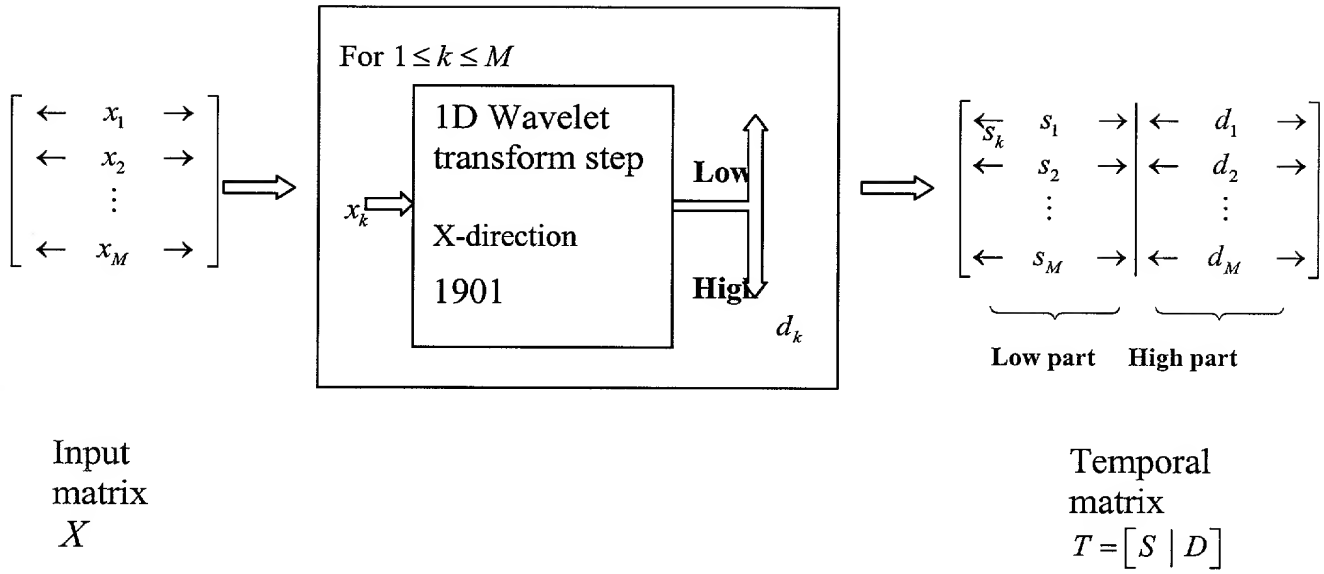
$$X=\begin{bmatrix} x(1,1) & x(1,2) & \cdots & x(1,N) \\ x(2,1) & x(2,2) & \cdots & x(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ x(M,1) & x(M,2) & \cdots & x(M,N) \end{bmatrix}$$

1801

2D Wavelet transform step

$$\hat{X}=\left[\begin{array}{c|c} LL & HL \\ \hline LH & HH \end{array}\right]=\left[\begin{array}{ccc|ccc} ll(1,1) & \cdots & ll(1,N/2) & hl(1,1) & \cdots & hl(1,N/2) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ ll(M/2,1) & \cdots & ll(M/2,N/2) & hl(M/2,1) & \cdots & hl(M/2,N/2) \\ \hline lh(1,1) & \cdots & lh(1,N/2) & hh(1,1) & \cdots & hh(1,N/2) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ lh(M/2,1) & \cdots & lh(M/2,N/2) & hh(M/2,1) & \cdots & hh(M/2,N/2) \end{array}\right]$$

Figure 18



1901

Figure 19

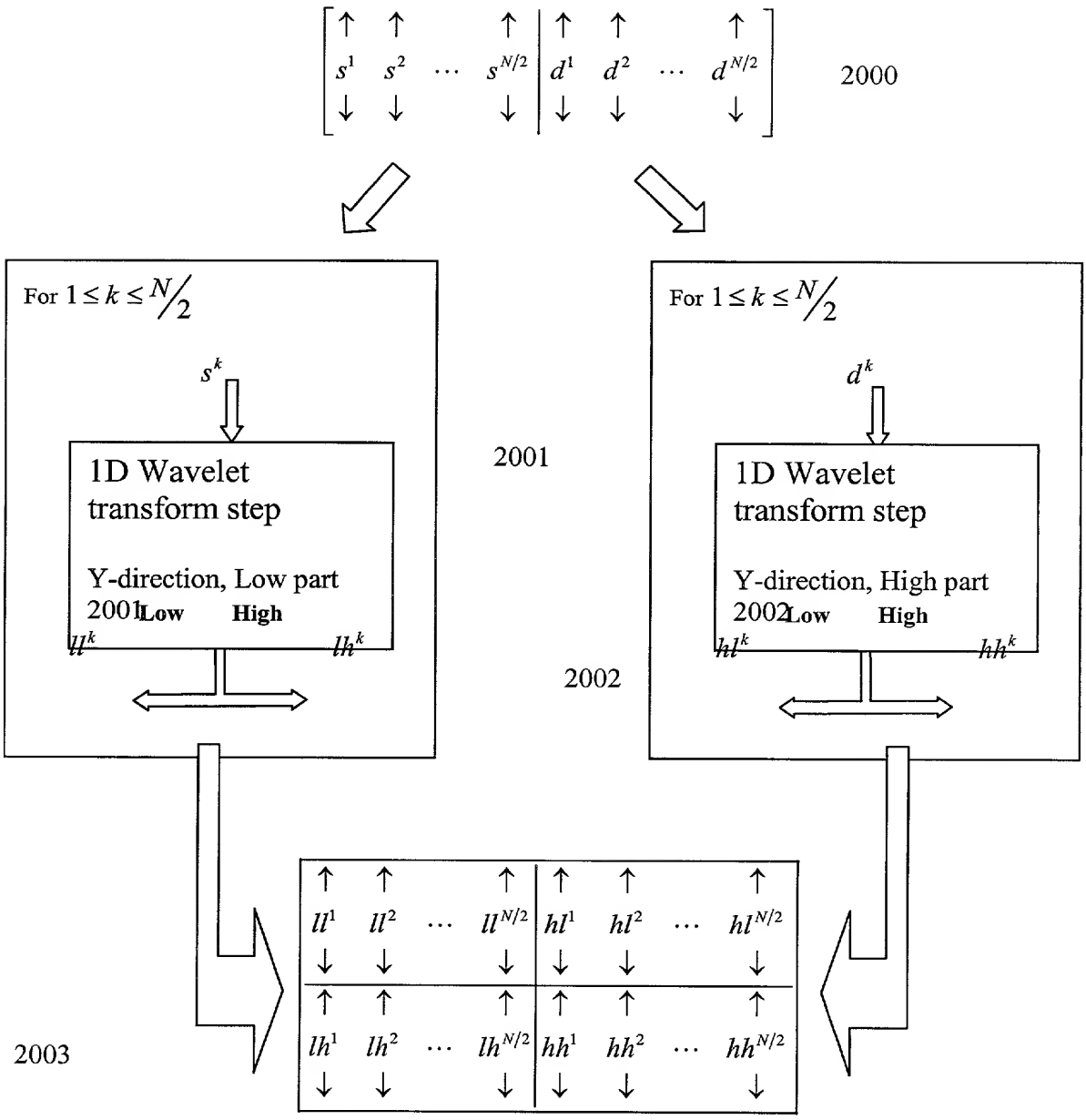

2000
For $1 \le k \le N/2$
s^k
1D Wavelet transform step
Y-direction, Low part
2001
Low
High
ll^k
lh^k
For $1 \le k \le N/2$
d^k
1D Wavelet transform step
Y-direction, High part
2002
Low
High
hl^k
hh^k
2003
ll^1 ll^2 ... $ll^{N/2}$ hl^1 hl^2 ... $hl^{N/2}$
lh^1 lh^2 ... $lh^{N/2}$ hh^1 hh^2 ... $hh^{N/2}$
Output matrix
2004
$\hat{X} = \left[\begin{array}{c|c} LL & HL \\ \hline LH & HH \end{array}\right]$

Figure 20

Let I be the original Image,

$X_0 = I$ ⟹ 2D Wavelet transform step ⟹ $\left[\begin{array}{c|c} LL_0 & HL_0 \\ \hline LH_0 & HH_0 \end{array}\right]$

For $0 < i < Levels$

$X_i = LL_{i-1}$ ⟹ 2D Wavelet transform step ⟹ $\left[\begin{array}{c|c} LL_i & HL_i \\ \hline LH_i & HH_i \end{array}\right]$

2100

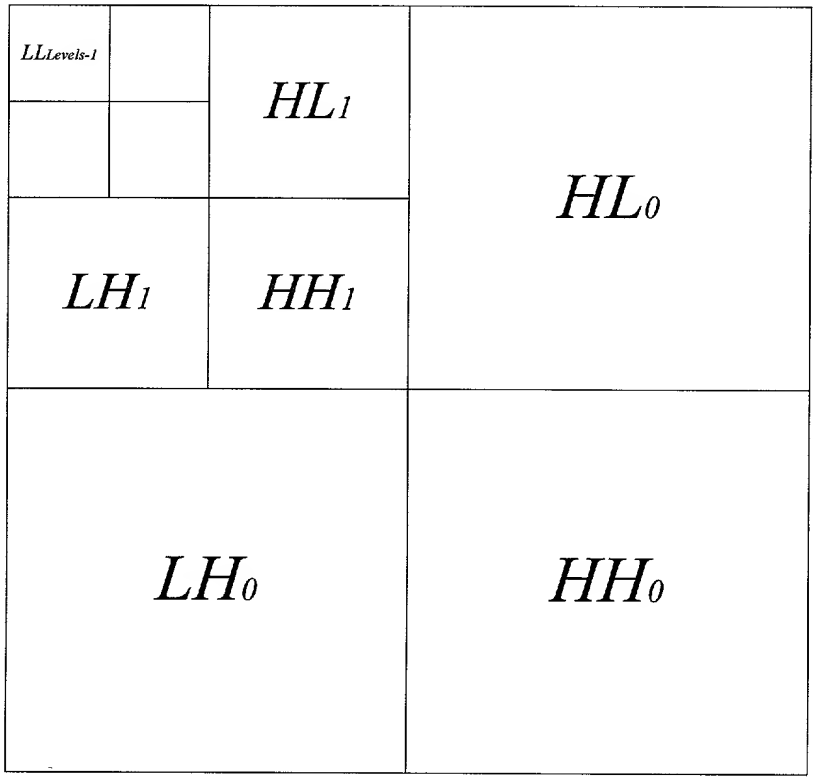


Figure 21

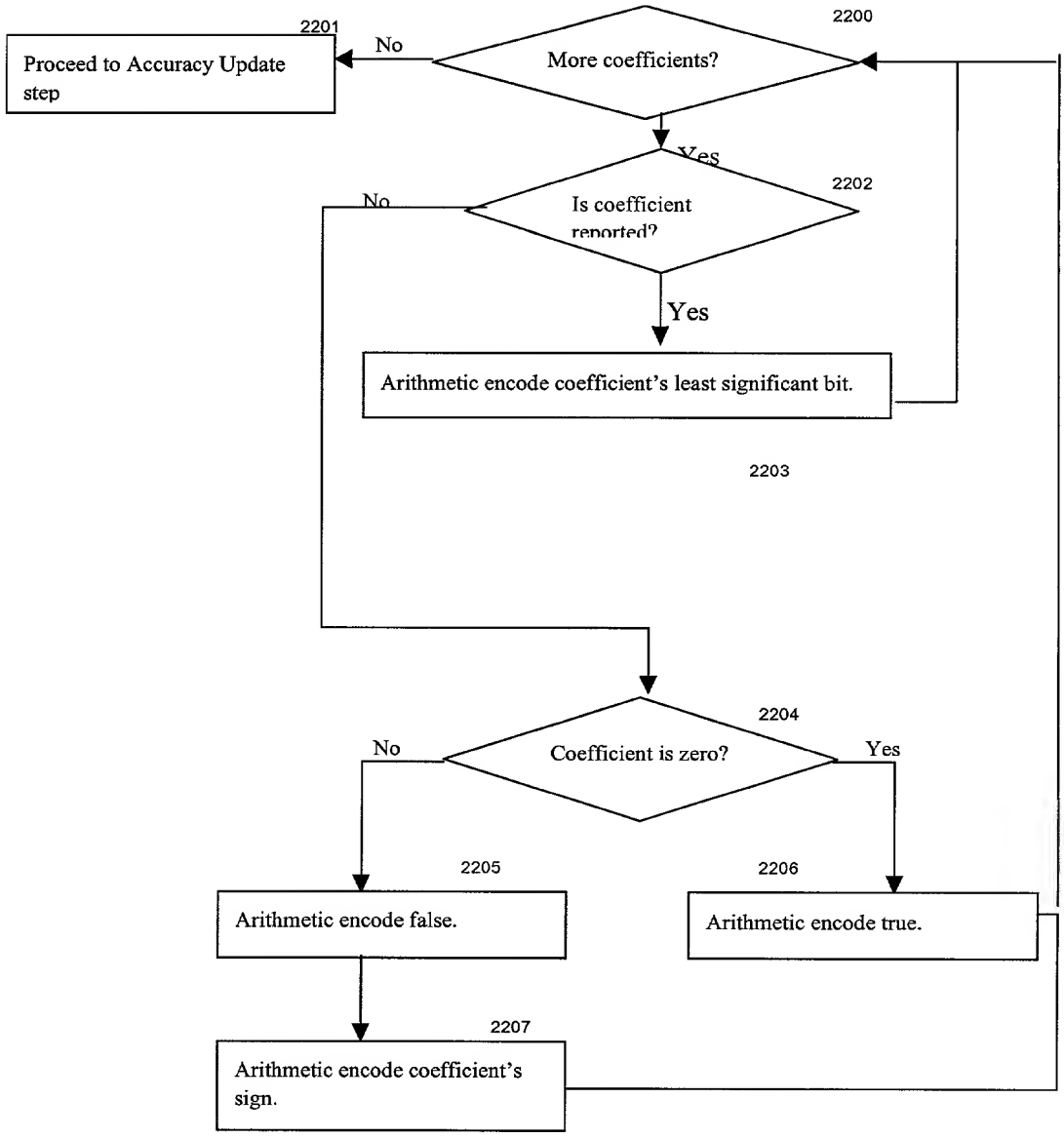

2200
More coefficients?
No
2201
Proceed to Accuracy Update step
Yes
2202
Is coefficient reported?
No
Yes
Arithmetic encode coefficient's least significant bit.
2203
2204
Coefficient is zero?
No
Yes
2205
Arithmetic encode false.
2206
Arithmetic encode true.
2207
Arithmetic encode coefficient's sign.

Figure 22

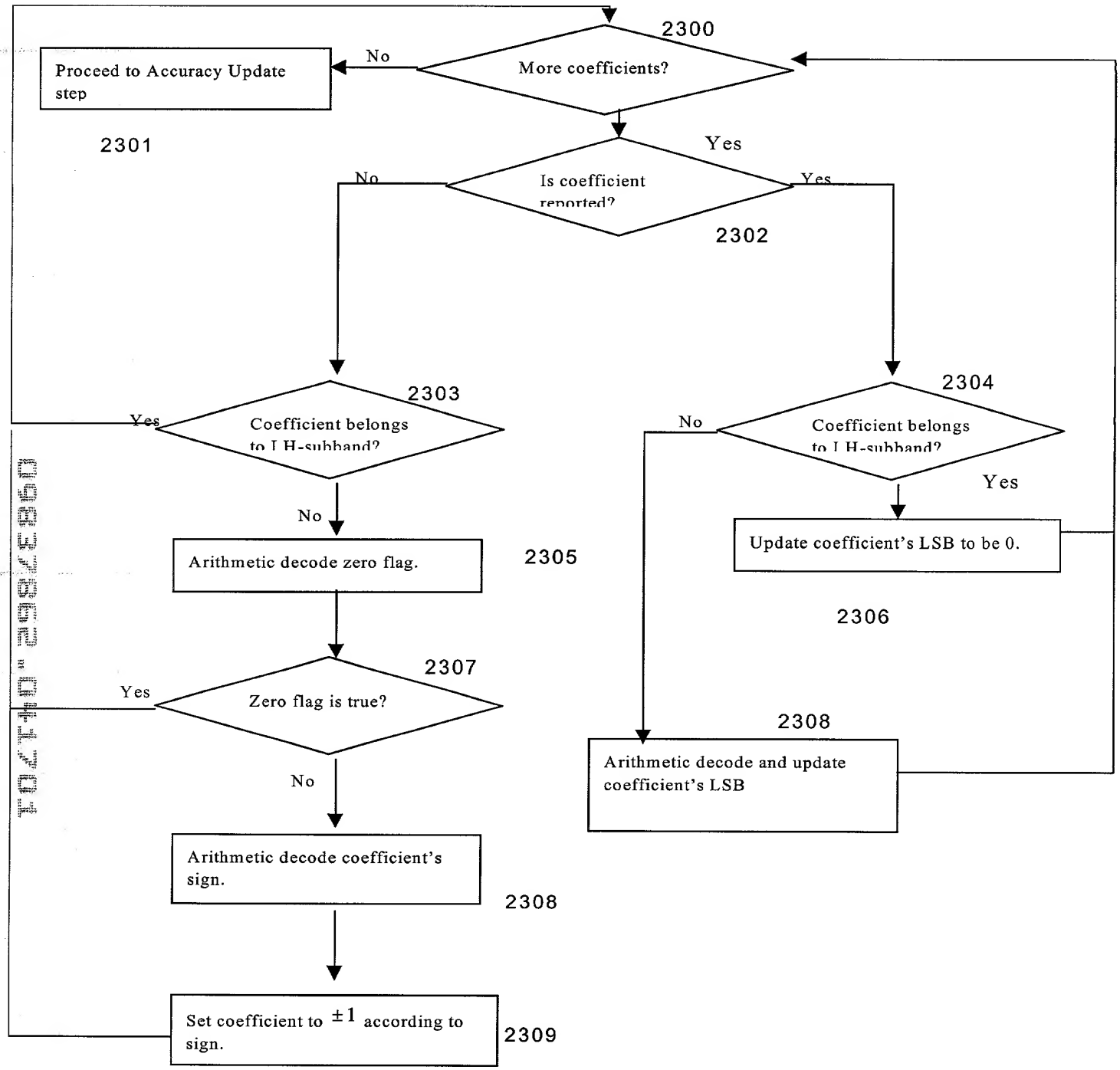

2300
More coefficients?
No
Proceed to Accuracy Update step
2301
Yes
Is coefficient reported?
No
Yes
2302
2303
Coefficient belongs to LH-subband?
Yes
No
Arithmetic decode zero flag.
2305
2307
Zero flag is true?
Yes
No
Arithmetic decode coefficient's sign.
2308
Set coefficient to ±1 according to sign.
2309
2304
Coefficient belongs to LH-subband?
No
Yes
Update coefficient's LSB to be 0.
2306
2308
Arithmetic decode and update coefficient's LSB

Figure 23

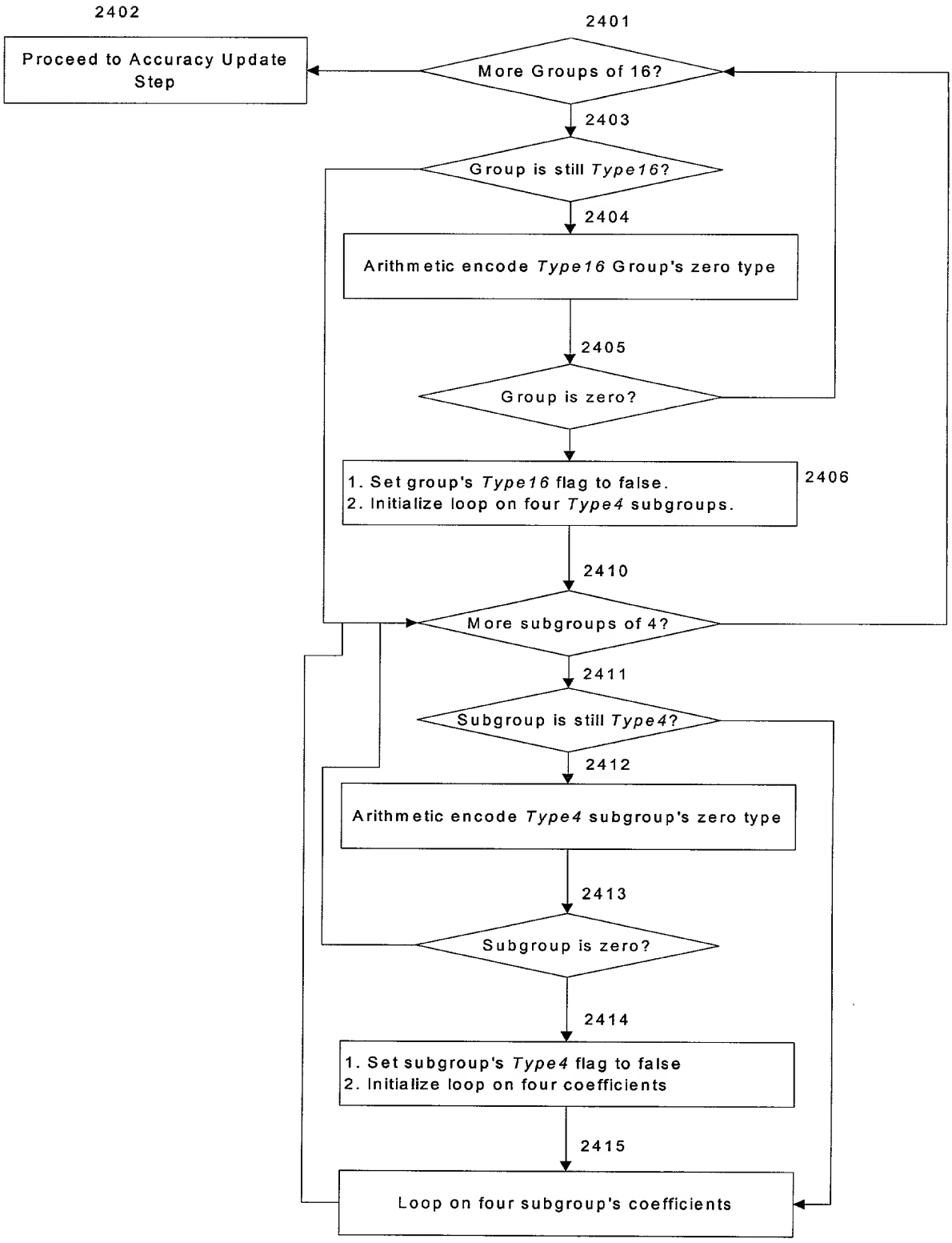

2402
Proceed to Accuracy Update Step
2401
More Groups of 16?
2403
Group is still Type16?
2404
Arithmetic encode Type16 Group's zero type
2405
Group is zero?
1. Set group's Type16 flag to false.
2. Initialize loop on four Type4 subgroups.
2406
2410
More subgroups of 4?
2411
Subgroup is still Type4?
2412
Arithmetic encode Type4 subgroup's zero type
2413
Subgroup is zero?
2414
1. Set subgroup's Type4 flag to false
2. Initialize loop on four coefficients
2415
Loop on four subgroup's coefficients

FIG. 24

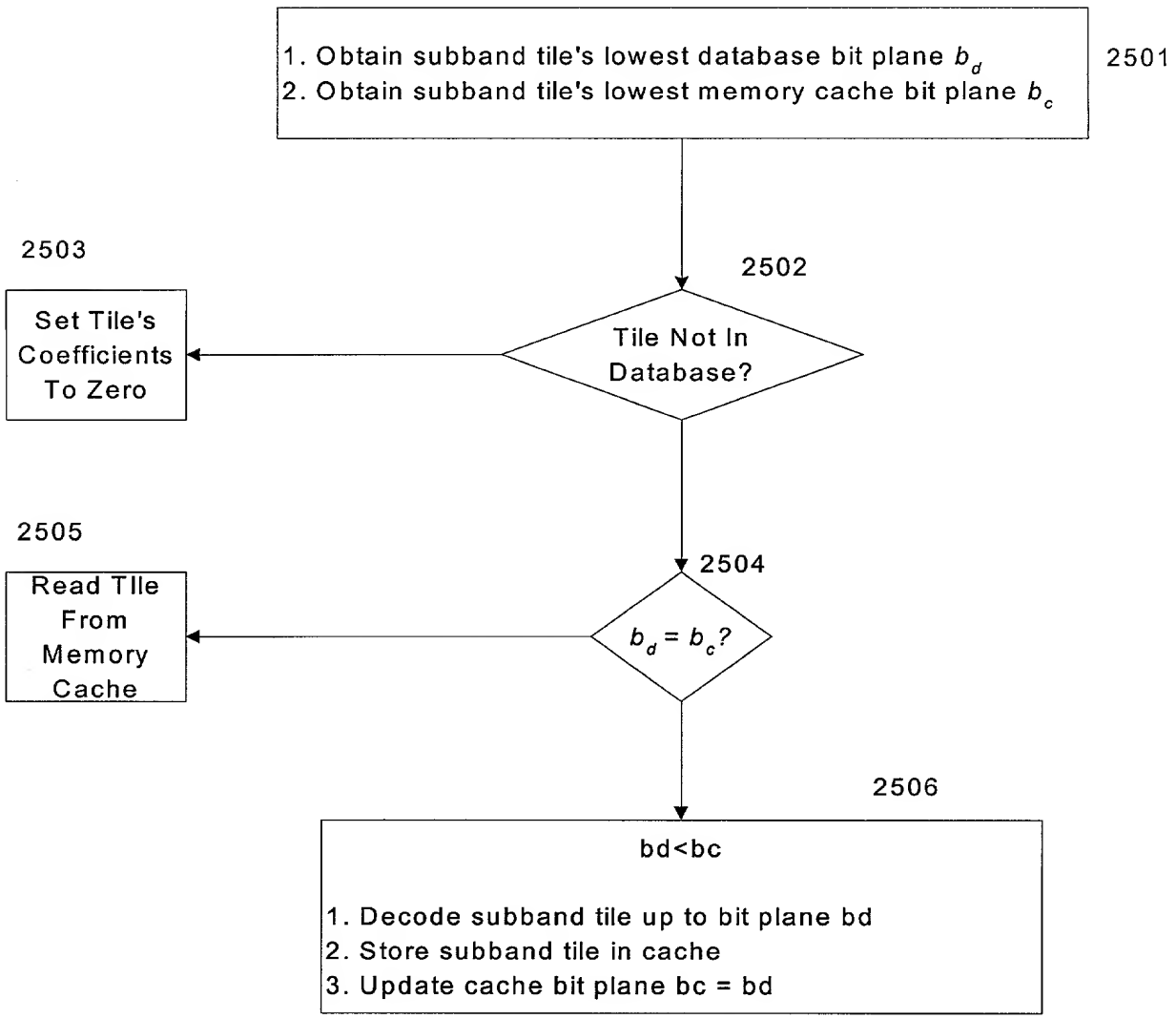

1. Obtain subband tile's lowest database bit plane b_d
2. Obtain subband tile's lowest memory cache bit plane b_c
2501
2502
Tile Not In Database?
2503
Set Tile's Coefficients To Zero
2504
$b_d = b_c$?
2505
Read TIle From Memory Cache
2506
bd<bc
1. Decode subband tile up to bit plane bd
2. Store subband tile in cache
3. Update cache bit plane bc = bd

Fig. 25

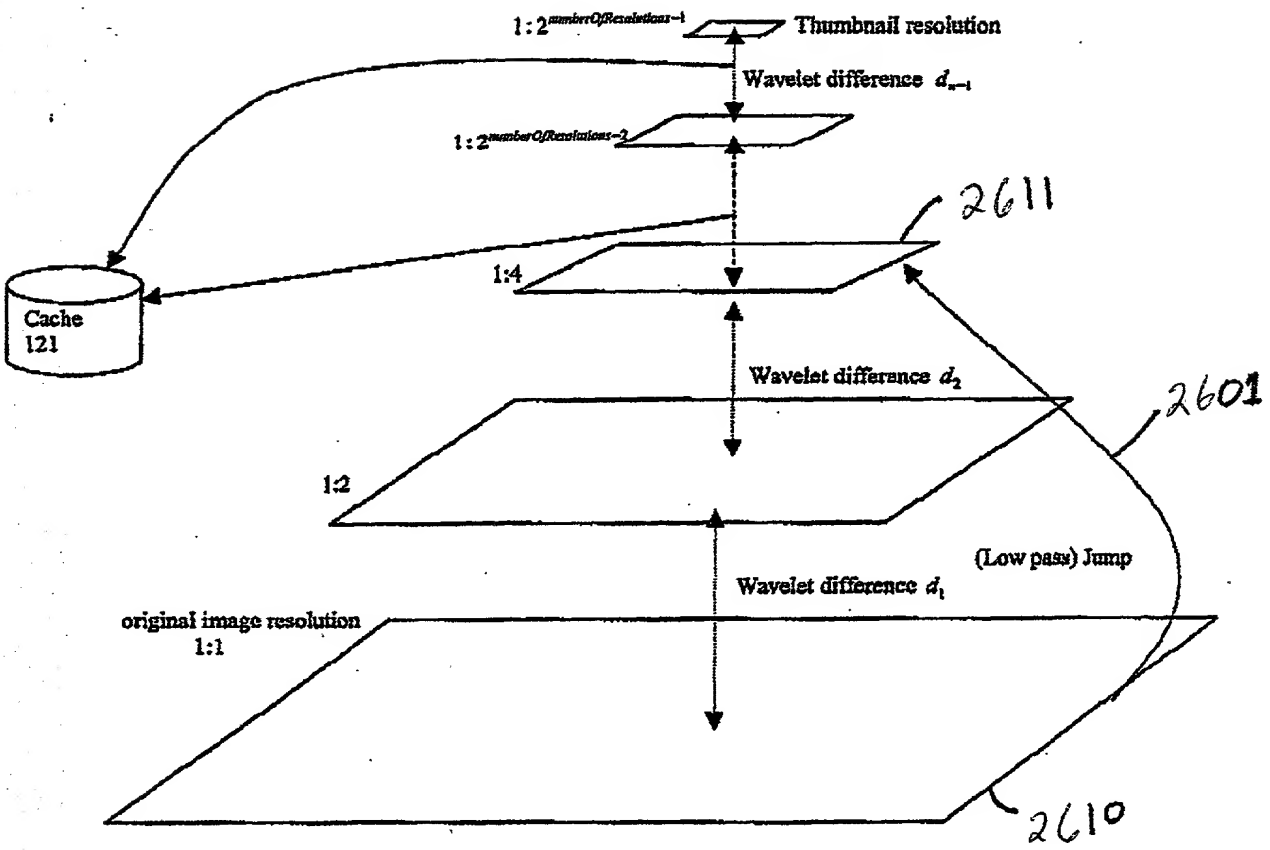


Figure 26 Preprocessing multiresolution structure